# Next Token Generation

Nipun Batra

July 26, 2025

IIT Gandhinagar

# Table of Contents

# Introduction and Motivation

## Acknowledgment and Inspiration

- This lecture is inspired by the excellent work of **Andrej Karpathy**

# Acknowledgment and Inspiration

- This lecture is inspired by the excellent work of **Andrej Karpathy**
- Search for *"Neural Networks: Zero to Hero"* to find his comprehensive tutorial series

# Acknowledgment and Inspiration

- This lecture is inspired by the excellent work of **Andrej Karpathy**
- Search for *"Neural Networks: Zero to Hero"* to find his comprehensive tutorial series
- These concepts are fundamental to understanding modern language models

## Acknowledgment and Inspiration

- This lecture is inspired by the excellent work of **Andrej Karpathy**
- Search for "*Neural Networks: Zero to Hero*" to find his comprehensive tutorial series
- These concepts are fundamental to understanding modern language models
- Direct relevance to **ChatGPT** and other large language models:

## Acknowledgment and Inspiration

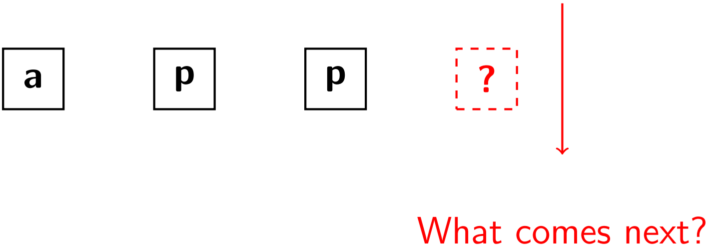
- This lecture is inspired by the excellent work of **Andrej Karpathy**
- Search for *"Neural Networks: Zero to Hero"* to find his comprehensive tutorial series
- These concepts are fundamental to understanding modern language models
- Direct relevance to **ChatGPT** and other large language models:
  - ChatGPT generates text by predicting the next token

# Acknowledgment and Inspiration

- This lecture is inspired by the excellent work of **Andrej Karpathy**
- Search for *"Neural Networks: Zero to Hero"* to find his comprehensive tutorial series
- These concepts are fundamental to understanding modern language models
- Direct relevance to **ChatGPT** and other large language models:
  - ChatGPT generates text by predicting the next token
  - Same underlying principle scaled to billions of parameters

## Acknowledgment and Inspiration

- This lecture is inspired by the excellent work of **Andrej Karpathy**
- Search for ”*Neural Networks: Zero to Hero*” to find his comprehensive tutorial series
- These concepts are fundamental to understanding modern language models
- Direct relevance to **ChatGPT** and other large language models:
  - ChatGPT generates text by predicting the next token
  - Same underlying principle scaled to billions of parameters
  - Understanding next token prediction is key to understanding LLMs




Given the sequence "app", what is the next character?

# Problem Formulation

# Next Character Prediction as Classification

**Classification Task:** Predict probability distribution over all

# Next Character Prediction as Classification

a

p

p

**Input: "app"**

**Classification Task:** Predict probability distribution over all

# Next Character Prediction as Classification

**Output: Character Probabilities**

**a**

**p**

**p**

**Input: "app"**

| Character | Probability |
|---|---|
| a | 0.05 |
| b | 0.02 |
| c | 0.03 |
| ... | ... |
| l | 0.35 |
| m | 0.01 |
| ... | ... |
| y | 0.08 |
| z | 0.01 |
| - (end) | 0.15 |

**Classification Task:** Predict probability distribution over all

# Case Study: Indian Names Generation

**Training Dataset**

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid

**Goal:** Learn to generate new, realistic Indian names

# Dataset: Indian Names

**Training Dataset**

- Abid
- Abhidha

**Goal:** Learn to generate new, realistic Indian names

# Dataset: Indian Names

**Training Dataset**

- Abid
- Abhidha
- Adesh

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya

**Goal:** Learn to generate new, realistic Indian names

### Training Dataset

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun

**Goal:** Learn to generate new, realistic Indian names

# Dataset: Indian Names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- $\vdots$

**Goal:** Learn to generate new, realistic Indian names

## Training Dataset

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- $\vdots$

**Goal:** Learn to generate new, realistic Indian names

# Dataset: Indian Names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- $\vdots$
- Kiran

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮
- Kiran
- Krishna

**Goal:** Learn to generate new, realistic Indian names

# Dataset: Indian Names

## Training Dataset

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮
- Kiran
- Krishna
- Lakshmi

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮
- Kiran
- Krishna
- Lakshmi
- Meera

**Goal:** Learn to generate new, realistic Indian names

# Dataset: Indian Names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- $\vdots$
- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- $\vdots$

- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha
- $\vdots$

**Goal:** Learn to generate new, realistic Indian names

# Dataset: Indian Names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮

- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha
- ⋮

**Goal:** Learn to generate new, realistic Indian names

## Dataset: Indian Names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮

- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha
- ⋮

- Priya

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮
- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha
- ⋮
- Priya
- Rajesh

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮
- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha
- ⋮
- Priya
- Rajesh
- Sunita

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮
- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha
- ⋮
- Priya
- Rajesh
- Sunita
- Vikash

**Goal:** Learn to generate new, realistic Indian names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮
- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha
- ⋮
- Priya
- Rajesh
- Sunita
- Vikash
- Zara

**Goal:** Learn to generate new, realistic Indian names

# Dataset: Indian Names

**Training Dataset**

- Abid
- Abhidha
- Adesh
- Aditya
- Arjun
- ⋮

- Kiran
- Krishna
- Lakshmi
- Meera
- Nisha
- ⋮

- Priya
- Rajesh
- Sunita
- Vikash
- Zara
- ⋮

**Goal:** Learn to generate new, realistic Indian names

# Assumptions and Constraints

- **Character Set:** Only 26 lowercase letters (a-z)

## Assumptions and Constraints

- **Character Set:** Only 26 lowercase letters (a-z)
- **End Marker:** Hyphen (-) indicates end of name

# Assumptions and Constraints

- **Character Set:** Only 26 lowercase letters (a-z)
- **End Marker:** Hyphen (-) indicates end of name
- **Length Constraint:** Names are between 4 and 10 characters

# Assumptions and Constraints

- **Character Set:** Only 26 lowercase letters (a-z)
- **End Marker:** Hyphen (-) indicates end of name
- **Length Constraint:** Names are between 4 and 10 characters
- **Simplification:** No spaces, special characters, or uppercase letters

# Assumptions and Constraints

- **Character Set:** Only 26 lowercase letters (a-z)
- **End Marker:** Hyphen (-) indicates end of name
- **Length Constraint:** Names are between 4 and 10 characters
- **Simplification:** No spaces, special characters, or uppercase letters

# Assumptions and Constraints

- **Character Set:** Only 26 lowercase letters (a-z)
- **End Marker:** Hyphen (-) indicates end of name
- **Length Constraint:** Names are between 4 and 10 characters
- **Simplification:** No spaces, special characters, or uppercase letters

**Vocabulary Size:**

26 letters $+$ 1 hyphen $=$ **27 characters**

# Training Data Generation

# Generate Training Dataset

**Example: ”abid” → Training Examples**

**Context Length:** 3 characters

| Input (X) | | | | Target (Y) |
|---|---|---|---|---|
| **Char 1** | **Char 2** | **Char 3** | **Context** | **Next Char** |
| - | - | - | ”—” | a |
| - | - | a | ”–a” | b |
| - | a | b | ”-ab” | i |
| a | b | i | ”abi” | d |
| b | i | d | ”bid” | - |

### Training Examples

From one name ”abid”, we create **5 training examples** using a

# Representation Learning

- **Goal:** Learn vector representations for each character

## The Idea: Character Embeddings

- **Goal:** Learn vector representations for each character
- **Hypothesis:** Similar characters should have similar embeddings

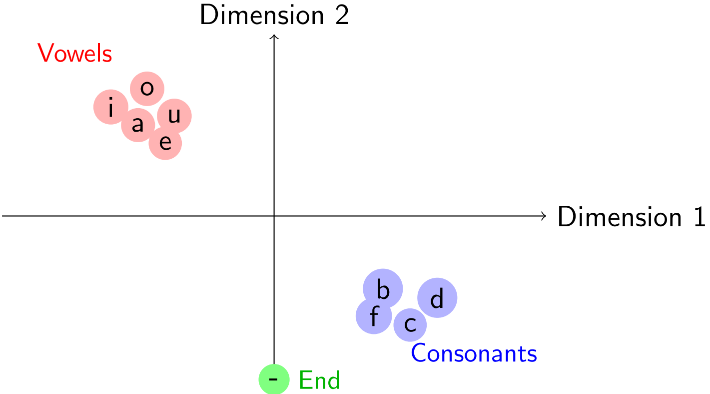
- **Goal:** Learn vector representations for each character
- **Hypothesis:** Similar characters should have similar embeddings
- **Benefit:** Capture semantic relationships between characters

## The Idea: Character Embeddings

- **Goal:** Learn vector representations for each character
- **Hypothesis:** Similar characters should have similar embeddings
- **Benefit:** Capture semantic relationships between characters

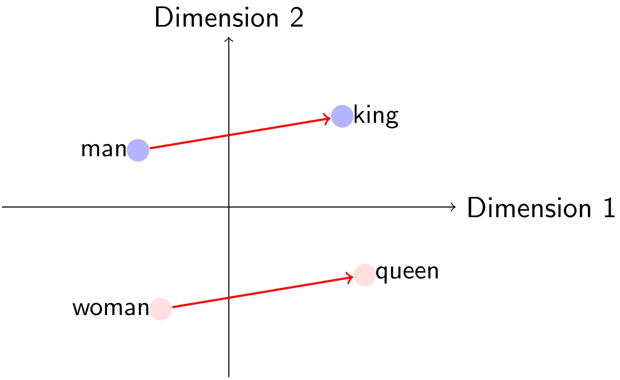
## The Idea: Character Embeddings

- **Goal:** Learn vector representations for each character
- **Hypothesis:** Similar characters should have similar embeddings
- **Benefit:** Capture semantic relationships between characters



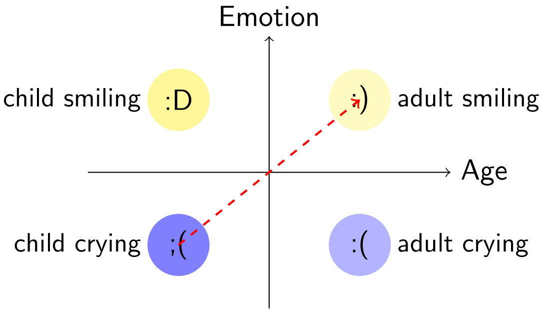
# Word2Vec Analogy Example

**Classic Word2Vec Relationship**




**Relationship:** queen $\approx$ king - man + woman

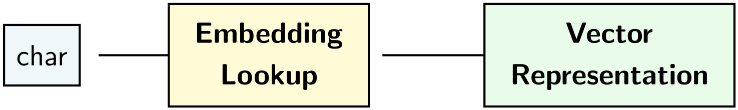
**Emotional Expression Analogy**




**Relationship:** child crying = child smiling + adult crying - adult smiling

# Embedding Architecture

# Embedding Matrix/Table Concept




**Process:** Character $\rightarrow$ Lookup in Embedding Table $\rightarrow$ Dense Vector

# Embedding Table Structure

**27 × K Embedding Matrix**

| Character | Dim 1 | Dim 2 | ... | Dim K |
|---|---|---|---|---|
| a | 0.2 | -0.1 | ... | 0.8 |
| b | -0.3 | 0.5 | ... | -0.2 |
| c | 0.1 | 0.3 | ... | 0.4 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| z | 0.7 | -0.4 | ... | 0.1 |
| - | 0.0 | 0.9 | ... | -0.5 |

**Key Point**

Each character maps to a K-dimensional dense vector representation.

# Learnable Parameters

- **Embedding Matrix:** 27 $\times$ K parameters

# Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
  - Initially random

# Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
  - Initially random
  - Updated during training via backpropagation

# Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations

## Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters

## Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
    - Initially random
    - Updated during training via backpropagation
    - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
    - Transform concatenated embeddings to output

# Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
    - Initially random
    - Updated during training via backpropagation
    - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
    - Transform concatenated embeddings to output
    - Learn classification patterns

# Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
  - Transform concatenated embeddings to output
  - Learn classification patterns
- **Total Learnable Parameters:**

# Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
  - Transform concatenated embeddings to output
  - Learn classification patterns
- **Total Learnable Parameters:**
  - Embedding: $27 \times K$

## Learnable Parameters

- **Embedding Matrix:** $27 \times K$ parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
  - Transform concatenated embeddings to output
  - Learn classification patterns
- **Total Learnable Parameters:**
  - Embedding: $27 \times K$
  - MLP: $(\text{context\_size} \times K) \rightarrow \text{hidden} \rightarrow \ldots \rightarrow 27$

# Neural Network Architecture

**Embedding Matrix (27 $\times$ 2)**

**Input: X = ["a", "b", "i"]**

| | D1 | D2 |
|---|---|---|
| a | 0.2 | -0.1 |
| b | -0.3 | 0.5 |
| … | … | … |
| i | 0.1 | 0.3 |
| … | … | … |
| z | 0.7 | -0.4 |
| - | 0.0 | 0.9 |

[0.2, -0.1]

[-0.3, 0.5]

[0.1, 0.3]

# Concatenate the Embeddings

**Feature Vector Construction**

a: [0.2, -0.1]

b: [-0.3, 0.5] $\xrightarrow{\text{concatenate}}$ [0.2, -0.1, -0.3, 0.5, 0.1, 0.3]

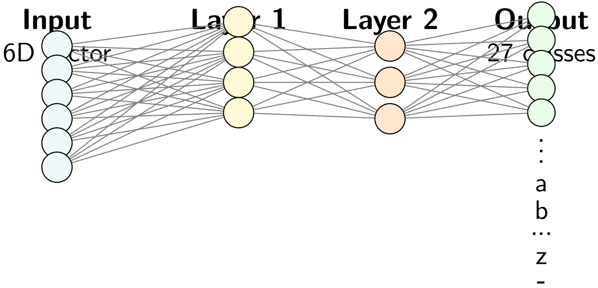
i: [0.1, 0.3]

**6-dimensional feature vector**

## Result

Context of 3 characters $\times$ 2D embeddings $=$ 6-dimensional input to neural network

# Multi-Layer Perceptron Architecture



# Training and Loss Function

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \tag{1}$$

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \tag{1}$$

- **What we learn:**

## Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \tag{1}$$

- **What we learn:**
    1. **Embedding Matrix:** Character representations (27 $\times$ K parameters)

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \tag{1}$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ($27 \times$ K parameters)
  2. **MLP Weights:** Neural network parameters for classification

## Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \tag{1}$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations (27 $\times$ K parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \tag{1}$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ($27 \times$ K parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input $\rightarrow$ Embeddings $\rightarrow$ Concatenate $\rightarrow$ MLP $\rightarrow$ Probabilities

## Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \qquad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations (27 $\times$ K parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input $\rightarrow$ Embeddings $\rightarrow$ Concatenate $\rightarrow$ MLP $\rightarrow$ Probabilities
  2. Compute cross-entropy loss

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \tag{1}$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ($27 \times$ K parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input $\rightarrow$ Embeddings $\rightarrow$ Concatenate $\rightarrow$ MLP $\rightarrow$ Probabilities
  2. Compute cross-entropy loss
  3. Backward pass: Update both embeddings and MLP weights

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \tag{1}$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations (27 $\times$ K parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input $\rightarrow$ Embeddings $\rightarrow$ Concatenate $\rightarrow$ MLP $\rightarrow$ Probabilities
  2. Compute cross-entropy loss
  3. Backward pass: Update both embeddings and MLP weights
  4. Repeat for all training examples

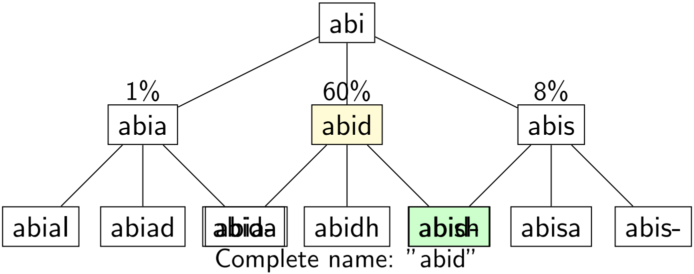
# Text Generation

**Test Input:** "abi"

**Predicted Probability Distribution**

| Next Char | Probability | Next Char | Probability |
|---|---|---|---|
| a | 0.01 | n | 0.05 |
| b | 0.01 | o | 0.02 |
| c | 0.03 | p | 0.01 |
| d | **0.60** | q | 0.00 |
| e | 0.02 | r | 0.03 |
| f | 0.01 | s | 0.08 |
| ... | ... | ... | ... |
| - | 0.05 | z | 0.01 |

# Generation Tree Structure




**Recursive Process:** Sample next character, append, repeat until end token

# Temperature and Sampling Strategies

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \tag{2}$$

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \tag{2}$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \tag{3}$$

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \tag{2}$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \tag{3}$$

- **Temperature Effects:**

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \tag{2}$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \tag{3}$$

- **Temperature Effects:**
  - $T = 1$: Standard probabilities

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \tag{2}$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \tag{3}$$

- **Temperature Effects:**
  - $T = 1$: Standard probabilities
  - $T \to 0$: More peaked (deterministic)

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \tag{2}$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \tag{3}$$

- **Temperature Effects:**
  - $T = 1$: Standard probabilities
  - $T \to 0$: More peaked (deterministic)
  - $T \to \infty$: More uniform (random)

# Temperature Variations

**Context:** "abi" $\rightarrow$ Next character probabilities

| **Character** | **T = 0.5 (Low)** | **T = 1.0 (Default)** | **T = 2.0 (High)** |
|---|---|---|---|
| a | 0.001 | 0.01 | 0.08 |
| d | **0.95** | **0.60** | **0.25** |
| s | 0.01 | 0.08 | 0.12 |
| h | 0.005 | 0.03 | 0.09 |
| - | 0.02 | 0.05 | 0.11 |
| others | 0.015 | 0.23 | 0.35 |

- **Low T:** More conservative, predictable names

# Temperature Variations

**Context:** "abi" $\rightarrow$ Next character probabilities

| Character | T = 0.5 (Low) | T = 1.0 (Default) | T = 2.0 (High) |
|---|---|---|---|
| a | 0.001 | 0.01 | 0.08 |
| d | **0.95** | **0.60** | **0.25** |
| s | 0.01 | 0.08 | 0.12 |
| h | 0.005 | 0.03 | 0.09 |
| - | 0.02 | 0.05 | 0.11 |
| others | 0.015 | 0.23 | 0.35 |

- **Low T:** More conservative, predictable names
- **High T:** More creative, diverse names

# Summary and Applications

## Key Takeaways

- **Core Idea:** Next token prediction as classification

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity

## Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling

## Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights

## Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models

## Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
  - GPT models use the same principle

## Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
    - GPT models use the same principle
    - Scaled to words/subwords instead of characters

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
  - GPT models use the same principle
  - Scaled to words/subwords instead of characters
  - Transformer architecture instead of MLP

## Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
  - GPT models use the same principle
  - Scaled to words/subwords instead of characters
  - Transformer architecture instead of MLP
  - Billions of parameters instead of thousands

## From Character-Level to ChatGPT

**Our Model**
Character-level
MLP
27 vocab
$\sim$1K params

$\xrightarrow{\text{Scale up}}$

**ChatGPT**
Token-level
Transformer
50K+ vocab
$\sim$175B params

**Same fundamental principle: Predict the next token!**